IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

**30 Years of GreatSPN**

**This is the author's manuscript**

*Original Citation:*

*Availability:*

This version is available http://hdl.handle.net/2318/1624717 since 2017-02-15T15:45:34Z

*Publisher:*

Springer International Publishing

*Published version:*

DOI:10.1007/978-3-319-30599-8_9

*Terms of use:*

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

06 May 2023

# 30 years of GreatSPN

Elvio Gilberto Amparore, Gianfranco Balbo, Marco Beccuti, Susanna Donatelli, Giuliana Franceschinis

**Abstract** GreatSPN is a tool for the stochastic analysis of systems modelled as (stochastic) Petri nets. This chapter describes the evolution of the GreatSPN framework over its lifespan of 30 years, from the first stochastic Petri net analyzer implemented in Pascal, to the current, fancy, graphical interface that supports a number of different model analyzers. This chapter reviews, with the help of a manufacturing system example, how GreatSPN is currently used for an integrated qualitative and quantitative analysis of Petri net systems, ranging from symbolic model checking techniques to a stochastic analysis whose efficiency is boosted by lumpability.

## 1 Introduction

GreatSPN [24] is a tool that supports model-based (stochastic) analysis of Discrete Event Dynamic Systems (DEDS). It has evolved significantly over its 30 years of life and it has been used not only for the evaluation of systems, but also to support research activities, by providing an environment suitable for the development of new methods and techniques, mainly aimed at performance evaluation.

The modelling formalisms of reference in GreatSPN are Generalized Stochastic Petri Nets (GSPN) [3] and its colored extension Stochastic Well-formed nets (SWN) [18]. SWN are based on the high-level Petri net model of well-formed net (WN). WN have been recasted into Symmetric Nets (SN) in the Petri net ISO/IEC 15909-2 standard [29], and therefore SWN are sometimes also called Stochastic Symmetric Nets (SSN).

GreatSPN was conceived about 30 years ago as a tool for performance evaluation. To overcome the (at that time) existing limitations in expressing synchronization and resource acquisition, GreatSPN evolved into a tool with a more holistic approach to verification. In this approach classical performance properties (like resource usage and throughput of transitions) and classical qualitative Petri net properties (like liveness of transition, existence of deadlocks and traps) and, more recently, probabilistic

verification properties, work in a synergic manner to establish the property of interest of a DEDS. In the rest of the paper we shall use the term "stochastic analysis" to refer to the set of analysis activities aimed at establishing the qualitative correctness of the model, using both performance and performability properties.

One of the distinctive features of GreatSPN with respect to other tools is the willingness of its development team to maintain, in the stochastic extension, the basic semantics of transition enabling and firing as well as the relevance of the graphical information. The underlying Petri net formalism is that of place/transition nets with priorities and inhibitor arcs, which have a comprehensive graphical representation of the net behaviour. The idea was to try to diverge as little as possible from the underlying Petri net formalism, so as to be able to reuse all possible solution techniques available for classical (non-stochastic) Petri nets. This choice enhanced the anlytical power, but certainly decreased the modelling power, since certain modelling features, like queueing policy for places and marking dependencies on arcs, have never been included.

In this chapter we review 30 years of history of GreatSPN and discuss its current role for model-based analysis: we revisit how the graphical interfaces have evolved over the years, and we also revisit many of the advances in stochastic Petri net analysis and what is the current status in model-based stochastic analysis.

The chapter starts with a review of the GSPN formalism: its roots and its evolution (Section 2), followed by the history of the GreatSPN tool in Section 3. The rest of the paper shows the current value of GreatSPN for model-based stochastic analysis. The tool, as it is now, is presented in Section 4. Section 5 describes how GreatSPN3.0 supports the workflow of a model-based analysis of a target system: from model construction to validation through model-checking and evaluation using stochastic model-checking and standard performance evaluation techniques. A similar workflow is illustrated for the colored case (Section 6). The common reference example is inspired by the various flexible manufacturing system models available in the literature. The chapter ends with a literature survey of tools with similar characteristics (Section 7) followed by a summary of the status of GreatSPN3.0 and of its desirable future (Section 8).

## 2 From Petri nets to GSPN

Petri Nets (PN) [34] are a natural, simple, and powerful formalism aimed at the modelling of the information and control logics in systems with asynchronous and concurrent activities. In Stochastic Petri Nets (SPN) [32] all the transitions are assumed to fire with a random delay that is exponentially distributed. This feature enriches the analysis of a variety of systems by computing several quantitative indices on their efficiency (performance) and reliability. Usually this is achieved by automatically constructing a Continuous Time Markov Chain (CTMC) which reflects the behavior of the system and then applying the analysis methods available for this type of stochastic processes.

In models of real systems, it is often the case that a change of state occurs not only because there has been a completion of an activity which takes time, but also because there has been a change of some logical conditions, which may depend on the current state of the system in a rather intricate manner. These two types of events may have rather different durations, and modelling these type of systems with SPNs yields CTMCs with quite different transition rates, making the numerical analysis of the model very difficult.

Starting from the practical observations, Generalized Stochastic Petri Nets (GSPN) [3] were proposed. Immediate transitions were introduced to answer the need for events that happen in a very short time (actually zero), it was also chosen that immediate transitions have priority over timed ones (the transitions that fire after a non-negligible amount of time). Priorities were introduced to simplify the analysis, by splitting markings in "vanishing" markings (states in which at least one immediate transition is enabled and where therefore the net does not spend any time) and "tangible" markings (states in which only timed transitions are enabled and where the net does spend time). Soon after their introduction, GSPNs became very popular in the performance and reliability evaluation community. The reason for this unexpected success was probably due to three quite different reasons: the simplicity of the formalism, the presence and the role of immediate transitions, and the availability (not much later than the formalism definition) of a design and analysis tool.

GSPNS are based on very few (and simple) primitive constructs that with their precise semantics make the formalism easy to learn and apply to many interesting practical problems. Indeed, researchers with considerably different backgrounds found GSPNs easy to grasp and useful for quickly drawing and analyzing complex probabilistic models that would have been otherwise difficult to construct in a reliable manner. Despite the need for more powerful formalisms for a compact representation of complex real systems, the choice of keeping the formalism simple while delegating to a different modelling language (namely Stochastic Well Formed nets - SWNs) the burden of dealing with these possible additional complexities allowed many newcomers to become quickly acquainted with GSPNs without scaring them away with less intuitive definitions. On the other hand, researchers already familiar with the features of the basic formalism found quite interesting the possibility of using with little additional effort the more complex high-level extensions, as they realized that this additional complexity pays off when it is actually needed by the difficulty of the problem at hand.

Immediate transitions were originally included in GSPNs to allow a simple representation of very fast activities and logical choices. The extensive use of the formalism made soon clear that the structure of the underlying un-timed (autonomous) net could play an important role in allowing many more results to be derived from the model. Time scale differences captured with timed and immediate transitions were related with the concepts of visible and invisible transitions in Petri net theory. The priority of immediate over timed transitions led to the study of un-timed Petri net with different levels of priority. Drawing on these results, it became clear the danger of specifying "confused models", and thus the difficulty of constructing

"correct models" including sequences of immediate transitions. To help analysts in specifying "well behaving nets", the concept of extended conflict set was formalized and methods were developed to find this structure at the net level [17, 37]. In the analysis of GSPN models, immediate transitions are thus "pre-processed" to construct a reduced Embedded Markov chain defined on tangible markings only.

Even the simplest GSPN models that one can conceive are difficult to describe and analyze without the use of proper software tools. The development and the free distribution of such a tool to academic researchers was indeed a key factor in spreading the knowledge and the use of GSPNs within the performance and reliability research communities.

As mentioned before, the complexity of real systems often requires more powerful formalisms in order to build relatively simple models, where abstraction is the key element to understand highly intricate situations. Colored extensions of (G)SPNs have thus been proposed, allowing a more compact and parametric model representation with a more efficient analysis. Such analysis is based on the strong or exact lumpability conditions on Markov chains. In particular Stochastic Well-Formed Nets (SWN) [18] have a structured color syntax that enables an automatic discovery of the presence of behavioral symmetries, leading directly to a reduced state space and a corresponding lumped CTMC.

## 3 The history of GreatSPN

The development of the Generalized Stochastic Petri Net (GSPN) formalism [2] was motivated by the modeling power of SPNs, with their effectiveness and simplicity [4]. A prototype solver [1] was initially jointly developed by members of the Computer Science Department of the University of Torino and of the Electronics Department of the Politecnico of Torino with the simple aim of overcoming the tedious and error prone task of manually constructing the Markov chains underlying GSPN models. Starting from the insights gained from the experience of using this preliminary tool, it was decided to design and implement a complete framework for the modeling, verification, and solution of GSPN models. The first version of the framework [15], written in Pascal [30], was released in 1985, and targeted three platforms: the VAX 11/780 with VMS, the VAX 11/780 with BSD Unix 4.1 and Sun 1 workstation with BSD Unix 4.2. This was the first documented software package for the analysis of GSPN models [23, p. 29].

The structure of the framework was conceived as a collection of interacting tools. Figure 1 (taken from [15], pag 139) shows the structure of this original tool chain. Programs (represented as rectangles) communicate with each others using intermediate files (represented as circles). Each program is designed to solve a specific problem. The purpose of this tool-chain was the generation of the reachability graph, computation of steady-state and/or transient solution of the reduced Markov chain, and computation of result statistics. GSPN and DSPN (Deterministic and Stochastic Petri Nets with deterministic transitions) models were supported.

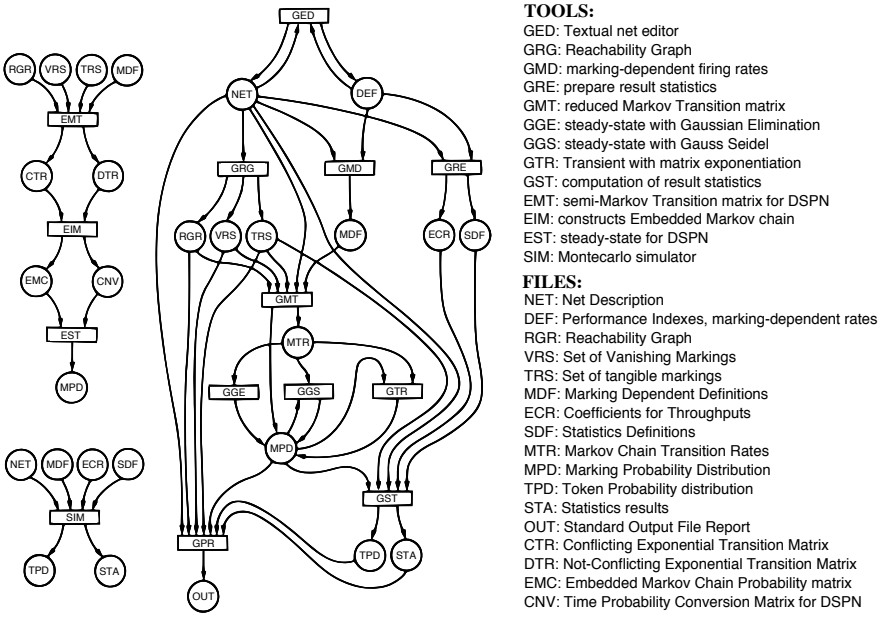



**Fig. 1** The structure of the GSPN/DSPN solution tool chain in 1985.

Started in August 1986 and based on the experience of Mike Molloy's SPAN interface [33] (which was the first graphical editor for SPN), a new GUI was written from scratch for the SunView 3.0 graphical toolkit, and its fusion with the GSPN solution tool-chain became the *GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets*, namely GreatSPN 1.0. GreatSPN was the first software package developed to fully integrate within a single user-friendly tool a modeling pipeline that included, among other features, editing Petri net models graphically, inspecting their properties like invariants and bounds, calling solvers, and showing graphically the results. In 1987, version 1.3 was released with all command line programs rewritten in C to increase the portability of the framework. Several "compiling techniques" [16] were introduced in this version to improve the time and space efficiency of the tool. With these new features, GreatSPN had the merit of joining a powerful graphical interface with a large variety of (both qualitative and quantitative) analysis methods. Models developed with GreatSPN showed for the first time the advantage of making practical the possibility for a performance analyst to first study qualitative properties of a systems with methods (s)he was little familiar with, and for formal method experts to complete their correctness and validation studies with performance considerations. Subsequently the addition of animation (token game) and discrete event simulation facilities [10] made GreatSPN beneficial also in management-oriented environments where the intuitive representations of real systems were more important than the powerful analysis methods that could be used for their evaluation. In 1995, the milestone release 1.7 introduced bound computa-

tion based on linear algebra techniques, as well as colored Petri nets in the form of *Stochastic Well-formed Nets* (SWN) [18]. The SWN solution method implemented in GreatSPN included both the state space generation and consequent solution of the associated Markov chain as well as the simulation, using either colored or symbolic markings.

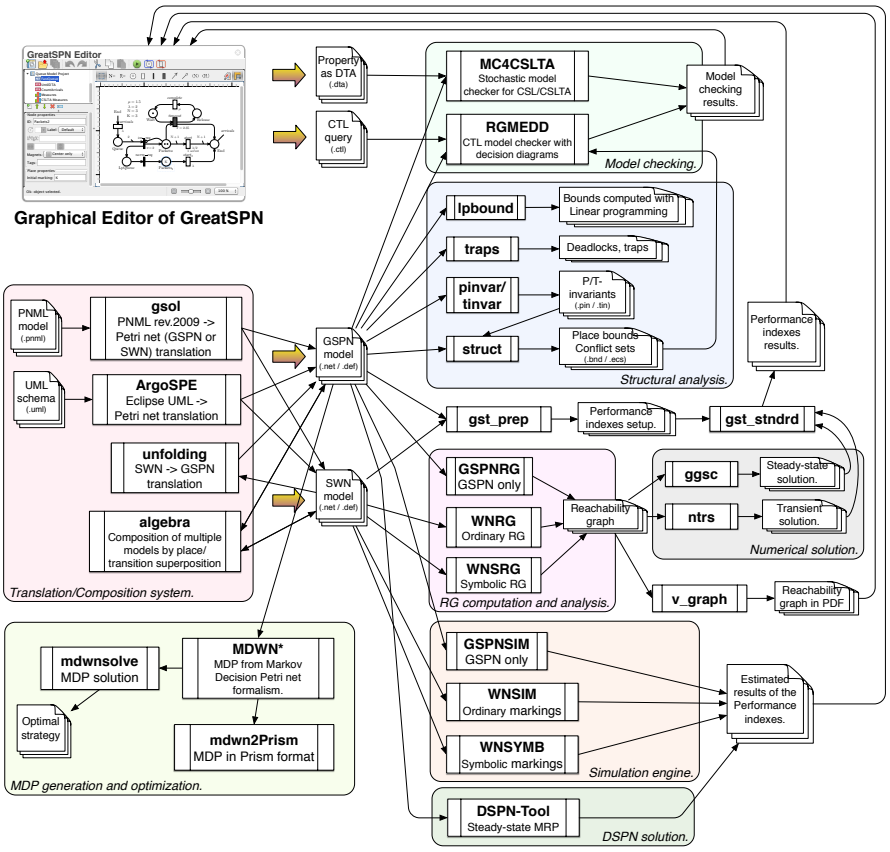



**Fig. 2** The structure of the GreatSPN framework today.

## 4 GreatSPN now

After almost 30 years of developments, improvements and tests, the GreatSPN framework is now a collection of many tools that support Petri net modeling and evaluation in multiple ways. Fig. 2 shows a (simplified) schema of the current features of GreatSPN. Tool names are written in bold, and are grouped into logical

modules. Tool functions include: numerical solutions, structural analysis, state space exploration and model-checking for GSPN and SWN, support for Markov Decision Well-formed nets (MDWN), conversions among multiple formalisms, Monte Carlo simulation, support for DSPN definition and solution, and model composition. The graphical editor is the center of GreatSPN as it is used for drawing the models and for defining their properties. It is responsible for the invocation of various command line tools and for the visualization of the results. GreatSPN is now in the transition of replacing the old Motif-based GUI with a new interface developed in Java. For the rest of the chapter, the characteristics of the framework will be shown from the user point of view, i.e. interacting with the new Java GUI. Most of the command line tools comprised in GreatSPN can be called directly from the GUI.

The workflow of GreatSPN was conceived, back in its original design, to consist of three main phases: the user ("modeler") draws the Petri net in a graphical way; Then structural properties are computed (minimal P/T semi-flows, place bounds, conflict sets, ...) to understand if the model is designed properly and if it can be solved using numerical methods or via simulation; Finally the user specifies the measures of interest directly on the model and calls a command line solver to compute the results. Several solvers are provided for different types of models and with different characteristics. Models are written to the disk in the net/def format, which contains the net description and the evaluation indexes to be computed. There are three families of models supported by GreatSPN: colored GSPNs, GSPNs with deterministic and/or general transitions, and Markov Decision Petri nets (MDWNs).

The new GUI [6] integrates a modern editing pipeline which supports the entire GreatSPN workflow consisting of the editing phase, the visual inspection of net properties, the evaluation of qualitative and quantitative properties, and visualization of the results. The rest of the chapter describes this implementation discussing a case study represented by a sufficiently complex GSPN model, that illustrates the details and the new features of the framework.

A picture of the new GreatSPN GUI is shown in Figure 3, taken while editing a Petri net model. In the upper-left panel, there is the list of open files. The editor supports *multi-page* file. In the current version of the editor, pages can be of three types: Petri net models, Deterministic Timed Automaton models (to be discussed later) and tables of measures. New model formalisms can be added to the editor by specifying new types of pages. The property panel is in the lower-left corner. It shows the editable properties of the selected objects. It is possible to operate to more than one object, of the same type, at a time. The central canvas contains the editor of the selected project page, in this case a GSPN model.

Petri nets are drawn with the usual graphical notation. Transitions may be immediate (thin black bars), exponential[1] (white rectangles) or general[2] (black rectangles). The priority level of immediate transitions is indicated as $\pi = i$ (omitted when $i = 1$). Input and output arcs are arrows and inhibitor arcs are circle-headed arrows. Arcs may be "broken", meaning that only the beginning and the end of the arrows

[1] Firing times are random variables with negative exponential distributions

[2] Firing times are random variables with general distributions

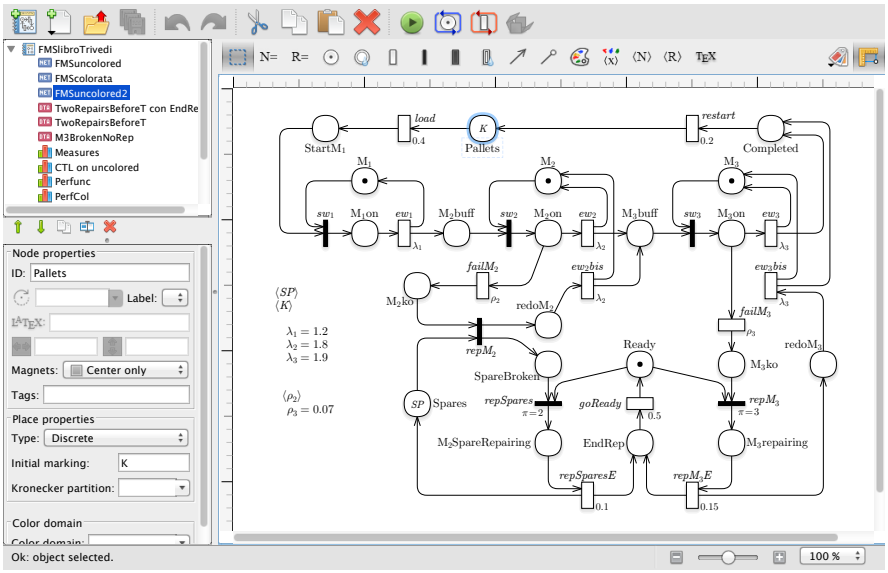



**Fig. 3** The GUI with a GSPN model of an FMS with faults and repairs.

are shown. The real values (or real-valued parameters) associated with transitions represent either the rate of the exponential distribution associated with timed transitions or the weight used to derive firing probabilities of immediate transitions. Definitions (constants, color classes, color variables) are drawn in textual form. Names, arc multiplicities, transition delays, weights, and priorities are all drawn as small movable labels positioned next to the corresponding Petri net elements.

Unlike the previous graphical interface, the check of the syntax of the colored definition is done while the definition is written. The editor also supports fluid places and fluid transitions (not shown in the example). Places can be partitioned into labeled groups for Kronecker-based solutions [14]. The editing process supports all the common operations of modern interactive editors, like undo/redo of every action, cut/copy/paste of objects, drag selection of objects with the mouse, single and multiple editing of selected objects, etc. Petri net models are drawn entirely using vector graphics, which allows for high quality visualization and print of the net. Object labels may be drawn with an optional LaTeX engine. The interface is designed to avoid modal dialog windows as much as possible to streamline the use of the GUI.

## 5 Model-based analysis through GSPN in GreatSPN3.0

Model-based analysis is supported by GreatSPN3.0 in various ways. The goal is to verify the correct behaviour of the modelled system through qualitative analysis and model-checking as well as to verify performance properties once it is decided that

the model correctly represents the system under study (as in the case of nets automatically generated from system specifications). Once the user is confident that the model represents the system behaviour, the analysis work-flow concentrates on the probabilistic aspects, through stochastic model-checking and through the computation of classical performance and/or dependability evaluation indices. The GUI of the tool supports the computation and the visualization of the results for a varying set of parameter values. The analysis workflow is illustrated on a rather classical GSPN model of a flexible manufacturing system (FMS). All the figures and the graphs reported are directly produced by GreatSPN3.0, unless otherwise stated.

The GSPN model of our FMS model is depicted in Figure 3, inside a screenshot of the GUI that has been used for its definition. The net by itself could also be printed as pdf file using the classical printing facilities of the operating system. The system includes three machines $M_i$ (places $\mathrm{M}_i$) and $K$ pallets of parts to be worked (place Pallets). Each part is loaded and then sequentially processed by the three machines until the work is completed, the manufactured part is unloaded and the pallet goes back to place Pallets through transition *restart* waiting for a new raw part to be loaded. For each machine $M_i$ an arriving part is taken (transition $sw_i$), worked (transition $ew_i$) and put in the input buffer of the subsequent machine or in the buffer of the completed parts. Machines $M_2$ and $M_3$ can fail. In the case of $M_2$ there are $SP$ spares available, while for $M_3$ there are no spare parts. Spares in use can fail as well. Both spares and machines are repaired by a repairman (token in place Ready). Since there are no spares for $M_3$ the repairman is assigned with higher priority to $M_3$. This is implemented through the priority of transition $repM_3$ which is higher than that of transition *repSpares*. Upon failure of $M_2$ (firing of transition $failM_2$), if no spare is available, the work to be done waits in place $\mathrm{M}_2$ko, while if a spare is available it is taken (transition $repM_2$), the machine goes into repair, and the piece is worked (transition $ew_2bis$). Finally the part is put in the input buffer of $M_3$ (place $\mathrm{M}_3$buff) and a token goes into the machine place $\mathrm{M}_2$ meaning that $M_2$ is available again. Upon failure of $M_3$, since there is no spare available, the part is blocked until the repair ends (transition $repM_3E$) and the part is worked (transition $ew_3bis$). Then machine goes back to $\mathrm{M}_3$ and the part goes into the buffer of completed parts.

The modeler can play the token game and observe the flow of tokens by firing the transitions. The token game can be driven by the modeler, which explicitly chooses which transition to fire among the set of enabled transitions, or can be delegated to the tool (random mode execution).

Step 1: standard qualitative properties

The analysis by P- and T-invariants, which can be activated from the graphical interface, reveals that there are 6 minimal P-semiflows and 4 minimal T-semiflows. The corresponding P-invariants prove that all places are bounded, with bounds equal either to $1, K$ or $SP$. Figure 4 shows one of the four T-semiflows as displayed by the GUI directly on the GSPN model. Transitions in the semi-flow are marked in red and their weight in the semi-flow is displayed inside the transition box (all weights

equal to 1 in this semi-flow). The T-semiflow of the figure refers to a pallet that goes normally through machines $M_1$ and $M_2$ and then experiences a failure at $M_3$. For this T-semiflow there is a firing sequence firable in the initial marking. The T-semiflow shows a scenario in which there is the direct intervention of the repairman to complete the work.

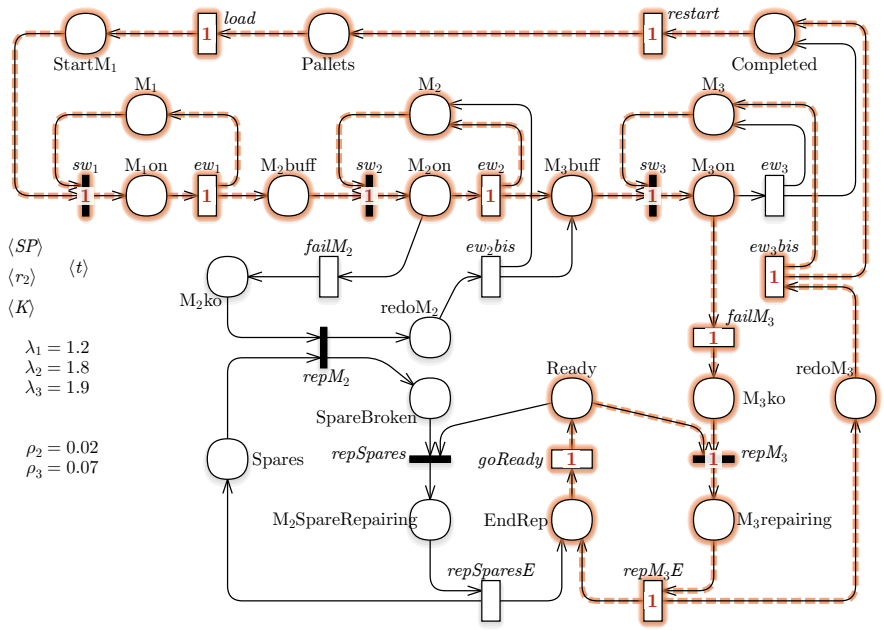



**Fig. 4** Visualization of a T-semiflow for the FMS model.

Reachability graph generation and analysis reveal that the state space contains a single connected component, that there are no deadlocks, and that all transitions are live. Reachability graphs can also be displayed by selecting the "measure" RG (TRG for the tangible reachability graph). Figure 5, right, shows the TRG as displayed by GreatSPN, while the left part is a zoom-in of a portion of it. The feature for displaying the reachability graph is very useful for Petri net beginners and for teaching, but it is usually not part of the normal work-flow of model-based analysis because the size of the graph when modeling realistic systems is most of the time too large to be conveniently visualized.

Step 2: Computational Tree Logic (CTL) model checking

More sophisticated properties of the net can be investigated through the CTL [20] model checker provided by the tool. The user defines one or more "CTL measures" in the measure panel, as shown in Figure 6. The analysis is performed for

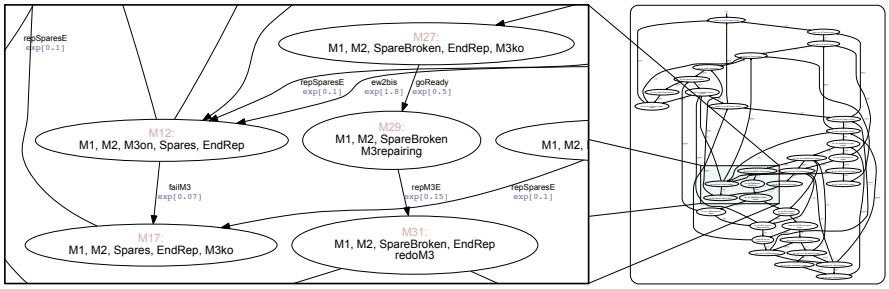



**Fig. 5** TRG visualization.

$SP = 3$, $K = 5$ (assigned in the top part of the window), since the GreatSPN model checker [5] is based on decision diagrams, much higher values of the parameters are verifiable. The CTL model checker of GreatSPN assigns a variable to each place. As usual in model-checkers based on decision diagrams, a bound on each variable should be known and a variable ordering should be defined. The central part of the window (Figure 6) is devoted to establish by which methods the bounds and the variable ordering have to be computed. The syntax of the CTL operators is the classical one, with A and E standing for "for all paths" and "there exists a path". Operators G and F stand for "for all states in the path" and "it exists a state in the path". The term $\#p$ means "number of token in place $p$", while condition $en(t)$ means "transition $t$ is enabled". The panel displays the truth value of each formula (computed in the initial state), but a log is available with more detailed information, including counter-examples or witnesses, whenever feasible. The properties listed in Figure 6 allow to investigate, from the top of the list downwards, more and more detailed aspects of the system behaviour.

The first point of the analysis is to check standard Petri net properties, like absence of deadlocks and liveness of transitions. Property 1° (AG ndeadlock) checks that on all states of all paths reachable from the initial marking (the whole RG) it is true that the state is not a deadlock. The panel of Figure 6 shows that the property is true and therefore the system has no deadlock. Property 2° (AG EF $en(load)$) is an example of liveness check. This property reads as "from all reachable states (AG) it is possible to find a path (EF) that enables *load*" ($en(load)$), which is equivalent to the classical definition of transition liveness in Petri nets. The screen-shot of Figure 6 shows that this property is true, so transition *load* is live.

Property 3° is instead an example of a model-dependent property, aimed at investigating the use of the spares. Do we really need all the *SP* spares that have been included in the model? Indeed property 3° checks if there is a reachable state in which there is a failure at $M_2$ and no spare is available. This property reads as follows: there is a state on a path (EF) for which machine $M_2$ has failed ( $\mathrm{M_2ko} > 0$) and no spare is available (Spares $= 0$). The property is true and a system designer may be tempted to add more spares to have a more efficient production process, but of course CTL analysis is not enough to assert how convenient this addition will be. This objective should be addresses by a quantitative (stochastic) analysis.

The fourth property investigates the need for the spares to be repaired (and for the repairman to get into action). This requirement has been translated into a CTL formula (4°:) that checks if it is true that on all reachable states (AG), on all paths that start from those states transition *goReady* is enabled (AF $en(goReady)$), that is to say, repairman goes back to the ready state. This property is false, since in the RG there are loops in which machines never break down; again, only a stochastic analysis can establish how often this happens, but it is well-known that, in the long run, the probability of having an execution in which machines never break down goes to zero. This is an instance of the classical fairness problem in CTL, which, when considering all paths, accounts also for the single (or the few), executions that will never happen under a fair schedule. GreatSPN3.0 is not able to check fair-CTL, but the modeler can use stochastic analysis to discriminate these situations, as we shall illustrate later in this section through the stochastic property of Figure 12. A stochastic approach might also be safer than a fair model checker: indeed in fair CTL the modeller explicitly indicates that the model checker should consider only the paths that verify certain conditions. If the modeler identifies the wrong conditions the whole analysis process can be severely impaired.

The last three properties (5° to 7°) verify how the system uses the spares (for example if spares are always taken one by one). 5° is an existential Until property that investigates whether there is a path from the initial marking in which the spares remain untouched ($\#Spares == SP$) until the count of spares is diminished by one. This property is true, but its value is limited. Indeed it does not say that there are no other paths in which a different behaviour is possible, not even that on that same path there could be a different behaviour. Property 5° is more informative since it checks that for all reachable states (AG) in which the number of spares is equal to *SP* all the paths stemming from that state keep *SP* spares until they get to $SP-1$. This property is false, and the tool provides a counterexample in the log file. A portion of this log is shown in Figure 7, which lists the states of a cyclic execution that starts with 3 spares and, passing through a number of reachable states, all with 3 spares, comes back to the initial state of the loop. Again, this is due to the presence of an infinite path in which machine $M_2$ never breaks down. Another line of investigation could be to check whether there exists a path, from the initial marking, in which all spares are available until two of them are taken in one step. This is formalized in property 7°, which is false.

Step 3: Classical performance evaluation

Once the user is confident that the model faithfully represents the system (at the desired level of abstraction), he/she can proceed to the standard performance index computation (throughput of transitions, average number of tokens in places and distribution of tokens in places, for various settings of the transition parameters and/or of the initial marking).

Figure 8 depicts two windows of the GreatSPN3.0 GUI. The measure panel (left) and a plot of the token distribution in place Pallets (right) which shows that most of

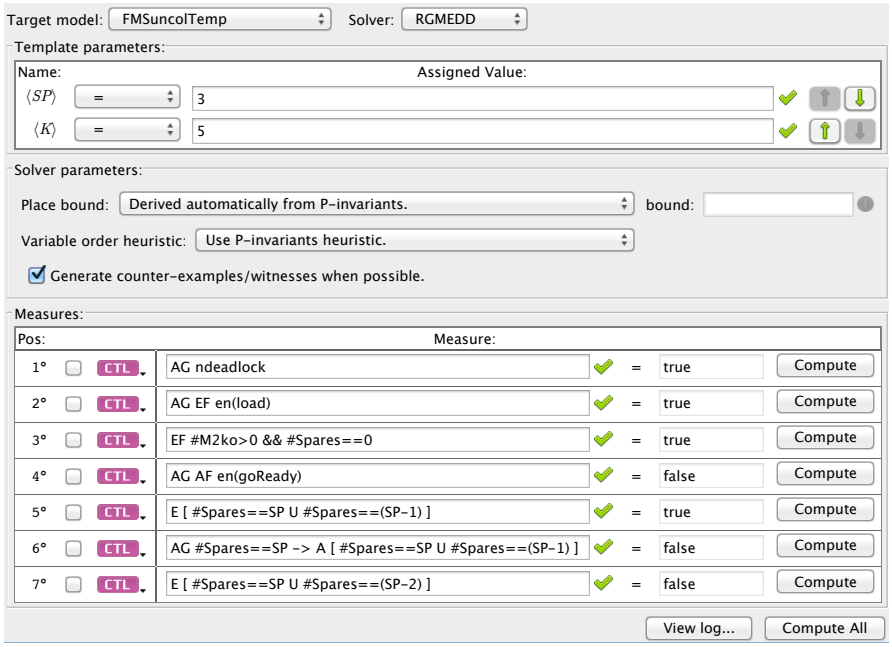



**Fig. 6** CTL model checker of GreatSPN.

```
1.1: Spares(3), Ready(1), Pallets(5), M3(1), M2(1), M1(1)
  State 1.1. does not satisfy: (#Spares == 2). Start of loop.

1.2: Spares(3), Ready(1), Pallets(4), StartM1(1), M3(1), M2(1), M1(1)
  State 1.2. does not satisfy: (#Spares == 2).

1.3: Spares(3), Ready(1), Pallets(4), M3(1), M2(1), M1on(1)
  State 1.3. does not satisfy: (#Spares == 2).

1.4: Spares(3), Ready(1), Pallets(4), M2buff(1), M3(1), M2(1), M1(1)
  State 1.4. does not satisfy: (#Spares == 2).

1.5: Spares(3), Ready(1), Pallets(4), M2on(1), M3(1), M1(1)
  State 1.5. does not satisfy: (#Spares == 2).

1.6: Spares(3), Ready(1), Pallets(4), M3buff(1), M3(1), M2(1), M1(1)
  State 1.6. does not satisfy: (#Spares == 2).

1.7: Spares(3), Ready(1), Pallets(4), M3on(1), M2(1), M1(1)
  State 1.7. does not satisfy: (#Spares == 2).

1.8: Spares(3), Ready(1), Pallets(4), Completed(1), M3(1), M2(1), M1(1)
  State 1.8. does not satisfy: (#Spares == 2).

1.9: loop back to state 1.1.
```

**Fig. 7** Log with a counterexample for property 6°

the time the place is empty. The measure panel is the same panel of Figure 6, where the target measure are performance indices instead of CTL properties. Measure ALL computes the (steady-state) token distributions for all the places and the throughput of all transitions, while specific performance indices (measure of type PERF can be defined with an appropriate grammar. Measure 3° defines the sum of the average number of tokens in the two places that represent machine $M_2$ correctly working, or of the two places that represents machine $M_2$ broken (measure 4°).

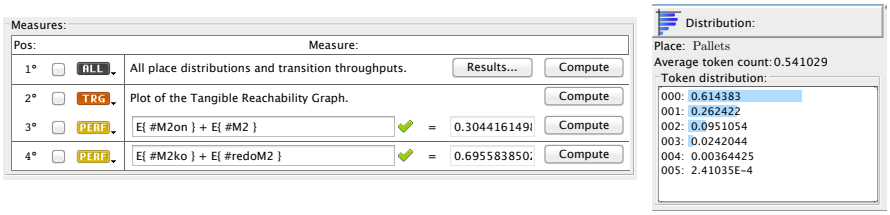



**Fig. 8** GUI for performance indices of the FMS and plot of the token distribution in place Pallets.

The tool also supports dependability measures like "how long machine $M_2$ will survive if no repair on $M_2$ can take place", for example for a varying number of parts circulating in the system. The condition that machine $M_2$ cannot be repaired is implemented in the model by simply removing transition *repSpares* (repair of a spare) so that machine $M_2$ and its spares are not repaired any longer. Figure 9 shows on the left the specified performance index (1°) and the range of variability of the parameters. In this case the analysis is conducted for 3 spares and a number of parts equal to $3,4$ and 5. The selected solution is the transient one at time $t$, where $t$ ranges in the interval $[4..40]$ with step 4. Measure 1° is the probability of finding all $SP$ spares in place SpareBroken. Results are plotted on the right side of Figure 9. The plot represents the distribution of the time needed to break all spares, for the three values of parameter $K$. The plot has been obtained using Excel on data exported through a specific GUI facility. Distribution of tokens in places and transition throughputs may also be shown and exported in Excel.

GreatSPN plots directly inside the GUI the distribution of the number of tokens in places (as in the case of the token distribution of Figure 8) and the throughput of transitions directly on the net elements in the net window of the GUI.

Step4: A "less classical" approach: performance and dependability through CSL$^{\text{TA}}$.

Standard and non-standard dependability/performance measures can be computed using the stochastic model checker for the CSL$^{\text{TA}}$[21] logic which is also part of the GreatSPN. A CSL$^{\text{TA}}$ formula has the form $\Phi = Prob_{\bowtie\alpha}(\mathscr{A})$, where $\bowtie$ is a comparison operator ($\bowtie \in \{\leq, <, =\}$), $\alpha$ a probability value and $\mathscr{A}$ is a timed automaton with a single clock $x$ that accepts/rejects timed paths of a GSPN. People familiar with simulation can think of the timed path of a GSPN as a simulation trace, with

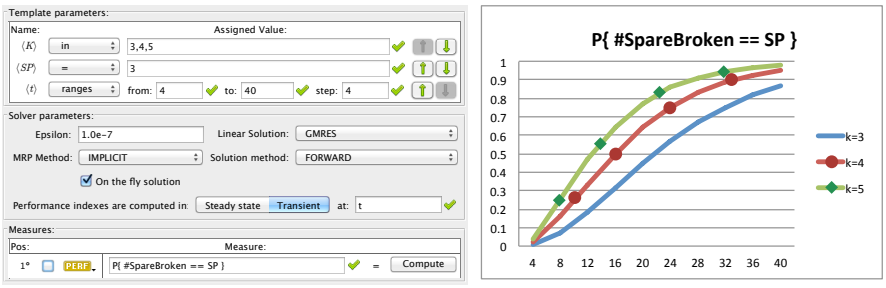


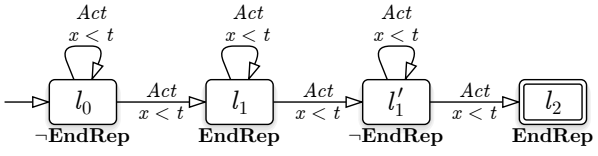

**Fig. 9** Distribution of the time for consuming all spares of machine $M_2$ for a varying number of parts.

states, events, and time at which events happen. A formula $\Phi$ is true in a marking $m$ if the probability of the set of the GSPN executions that starts in $m$ and that are accepted by $\mathscr{A}$ is $\bowtie \alpha$. Model checking algorithms for CSL$^{\text{TA}}$ exist that return true if the property is true for the initial marking. The CSL$^{\text{TA}}$ model checker of GreatSPN [7] computes the truth value for the initial marking as well as, for each reachable marking, the probability of the set of accepted executions stemming from that marking. The check of a CSL$^{\text{TA}}$ property has a cost which, in the worst case, is equivalent to the cost of solving in steady-state a Markov Regenerative Process.

Figure 10(left) shows the automaton that accepts the GSPN executions in which the repairman completes two repairs before time $t$. The automaton has four locations, including $l_0$, the initial one, and $l_2$, the accepting one. The timed automaton of a CSL$^{\text{TA}}$ formula can have more than one initial and more than one accepting locations. There is a single clock $x$ that starts from zero and increases linearly unless it is reset to zero. A condition $x \leq t$ on an arc implies that the arc can be taken only when the value of the clock $x$ is $\leq t$. Arcs are labelled with GSPN transition names (where *Act* stands for "any transition") and location have an associated property that is evaluated over the GSPN marking. If the clock $x$ is attached to the arc, taking the arc implies that $x$ is set to zero. When the GSPN moves from marking $m$ to marking $m'$ for the firing of $t$, this move is accepted by the automaton only if there is an arc labelled $t$ out of the current location, the clock guard is satisfied, and the arc leads to a location whose atomic proposition is satisfied by marking $m'$. The only other way in which an automaton can move is because of a condition $x = \beta$ associated to an arc from the current location of the automaton. When the clock reaches value $\beta$ the automaton "takes the arc" and changes location.

For a GSPN execution to be accepted by the automaton of Figure 10, it must start in a $\neg$**EndRep** marking, then move, in one of more transition firings, to an **EndRep** marking, then, back to a $\neg$**EndRep** marking. The execution is finally accepted when the GSPN goes back to an **EndRep** marking. Whenever the clock reaches $t$, the path is discarded. When the atomic proposition **EndRep** is instantiated to the GSPN condition "#EndRep $= 1$" and the value of $t$ is instantiated to 30, the CSL$^{\text{TA}}$ model checking algorithms computes the probability of the accepted paths. This probabil-

ity is listed in table reported in Figure 10 (right). Parameter $\rho$ is the rate of failure of both $M_2$ and $M_3$.



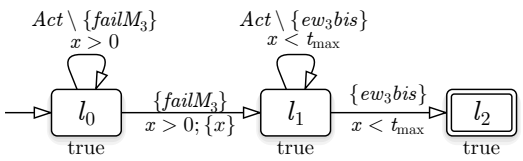

K=5, SP=3, t=30

| ρ | Probability |
|---|---|
| 0.05 | 0.081078 |
| 0.1 | 0.216116 |
| 0.5 | 0.653851 |
| 1.0 | 0.709993 |
| 5.0 | 0.737052 |
| 10.0 | 0.739829 |

**Fig. 10** Probability of two repairs, any machine, before time $t = 30$ for a varying failure rate $\rho$.

Figure 11 shows an example of a timed automaton used to compute the distribution of a completion time. The automaton accepts all executions that take less than $t_{max}$ from the first failure at $M_3$ (firing of $failM_3$) to completion of the part that underwent the machine failure (firing of $ew_3bis$). The analysis plotted in Figure 11 is for a varying value of $t_{max}$ from 5 to 50.

Note that the automaton in Figure 10 accepts paths depending only on the visited markings. since there are no specific requirements for actions associated to arcs, the automaton of Figure 11 only accepts a path based on the transitions that occurs along that path (the atomic proposition associated to location is always the clause "True").




K=5, SP=3

| Time | Probability |
|---|---|
| 5 | 0.300606 |
| 10 | 0.534543 |
| 15 | 0.692577 |
| 20 | 0.798302 |
| 25 | 0.868422 |
| 30 | 0.914588 |
| 35 | 0.944795 |
| 40 | 0.964454 |
| 45 | 0.977190 |
| 50 | 0.985406 |

**Fig. 11** Less than $t_{max}$ time units from first $M_3$ failure to completion of the part that underwent the machine failure.

There is another less common use of probabilistic verification that can be very useful. As previously discussed, there are certain CTL formulas which are false due to the presence of some anomalous behaviour, like infinite executions in which a machine never breaks down. Typically these executions are not realistic. Indeed the property "AGAF $en(goReady)$ was shown to be false. We can define a similar property in CSL$^{\text{TA}}$ through the automaton of Figure 12, which accepts all paths in which transition *goReady* fires at least once. The CSL$^{\text{TA}}$ model-checker reveals that, for all states, the set of paths accepted by the timed automaton has probability 1, clearly indicating that in a probabilistic setting the path(s) that makes the CTL formula false are negligible. This is indeed an example on the importance of having in a single tool *both* qualitative and probabilistic model checking.

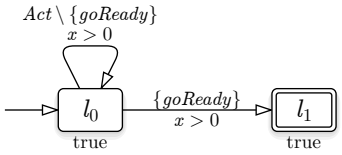



**Fig. 12** Probability of executing *GoReady* at least once

## 6 Model-based analysis through colored GSPN in GreatSPN3.0

GreatSPN supports also a formalism in the class of high level Petri nets, namely Stochastic Symmetric Nets[3] (SSN)[18]. The new GreatSPN GUI supports the design of SSN models as well as the interactive simulation with the *colored token game*. It also includes the *unfolding* function that generates a GSPN model whose behavior is equivalent to that of the SSN.

Figure 13 shows a variation of the FMS model, enriched with *colors*, and expressed through the SSN formalism. The model definition includes a set of finite basic color classes, and set of place color domains which result from the Cartesian product of basic color classes and define the possible *colors* of the tokens in each place. Transition color domains define the possible *color instances* of each transition. The arc functions define the multiset of tokens withdrawn from or added to the input and output places by a given transition instance. Colors may be useful in different situations, like when there is a need to identify a specific token among a set of tokens residing in the same places (e.g. to compute first passage time distributions [11]). This differentiates the qualitative behavior of some entities (hence making the marking evolution to depend on colors) or the stochastic delays of the activities involving specific entities (hence defining color dependent transition rates). In general colored models are more compact and can highlight symmetries in the model.

The SSN model in Figure 13 has two color classes, $C$ (identifiers of parts being processed by the FMS), partitioned into three static sublcasses: $C_1$, $C_2$ and $C_3$, and $CM$ (faulty machine identifiers: $m_2$ and $m_3$ corresponding to machines $M_2$ and $M_3$) containing two static subclasses, one for each of the machine that may breakdown and be repaired. The components in subclasses $C2$ and $C3$ skip the processing on machine $m_2$. This is modeled by the two guards associated with transitions $t0$ and $t1$, namely $[(x \in C2) \vee (x \in C3)]$ and $[(x \in C1)]$. The repair procedure for the two machines in this model is the same, and is based on the availability of spares. We shall consider two configurations: an asymmetric one where we have three spares for machine $m_2$ and only one for machine $m_3$, and a symmetric one, where there are two spares for each machine. There is also a repairman who replaces the broken spares of both machines. In the model we can observe that the broken spares of machine $m_3$ are repaired with priority over those of machine $m_2$ (indicated by the labels $\pi = 3$ and $\pi = 2$ next to transitions $repSp3$ and $repSP2$).

[3] The formalism was first introduced with the name of Well-Formed Nets, but recently it has been replaced by the new name Symmetric Nets, better emphasizing its specific features.

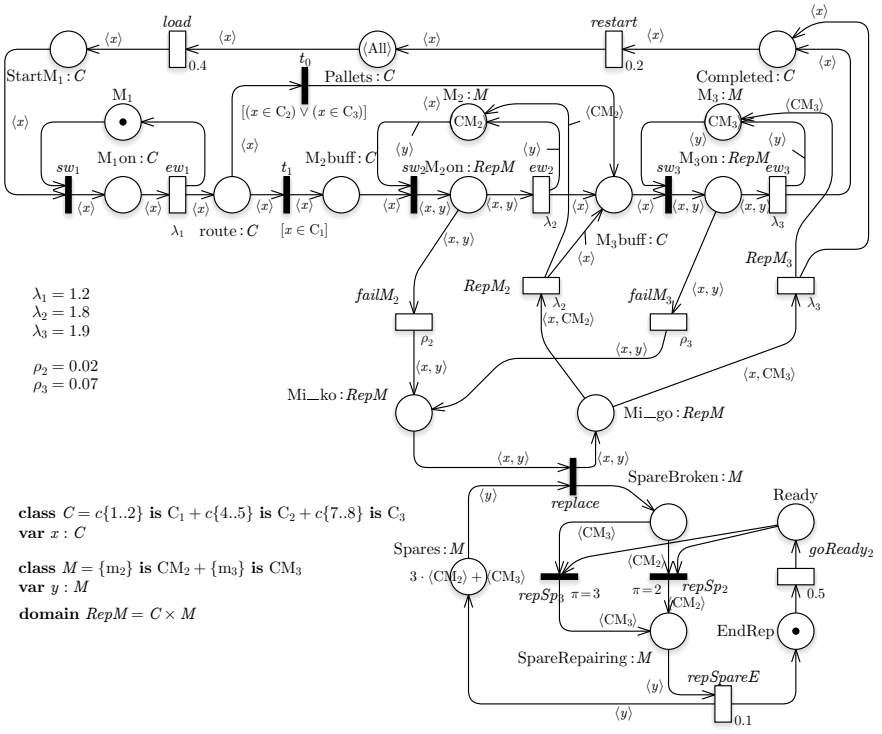



**Fig. 13** A colored model of an FMS with faults and repairs.

For an early check of the qualitative behavior of the colored model in the design phase, it is possible to use the colored token game feature of the GUI, which in any given marking highlights all the transitions that have at least one enabled instance. Clicking on one such transition a list of enabled instances pops up so that the modeler can choose which one to fire, leading to a new colored marking. The trace of markings reached along the interactive simulation is shown, and the user can return to one of the visited markings by clicking on it to try another path originating in that marking. During this initial phase the transition timing can be taken into account allowing the GUI to generate random delays to be associated with transitions enabled in a given marking.

The analysis of colored models can proceed in two directions: (1) through the *unfolding* of the colored net (now available with one click through the new GUI) into a (usually much larger) equivalent net without colors that can be processed using the solvers illustrated earlier, or (2) by direct analysis of the colored model using SSN specific solvers.

Table 1 shows the number of places and transitions in the unfolded version of model depicted in Fig. 13 for different sizes of color class *C*. The table reports also the sizes of the ordinary reachability graph (RG), and that of the symbolic reach-

| Configuration with asymmetric number of spares and color independent/dependent transition rates in $ew_3$ and $RepM3$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $C$ size | $\|SRG\|$ | | $\|RG\|$ | | $ESRG$ (Tan+Van) | | | Unfolded |
| $\|C1\|,\|C2\|,\|C3\|$ | Tangible | Vanishing | Tangible | Vanishing | $\|ESRG\|$ | $\|Strong\|$ | $\|Exact\|$ | $\|P\|$, $\|T\|$ |
| $2,2+2$ | 40 985 | 116 663 | 965 596 | 3 061 896 | 41 471 | 54 608 | 155 171 | 103,131 |
| $2,2,2$ | 159 786 | 484 140 | 965 596 | 3 061 896 | 41 471 | 61 057 | 617 185 | |
| dep.rates | | | | | 41 471 | 76 423 | 617 185 | |
| $3,2+2$ | 118 225 | 367 670 | 7 046 972 | 24 442 193 | 79 052 | 96 500 | 476 190 | 118,152 |
| $3,2,2$ | 455 230 | 1 491 364 | 7 046 972 | 24 442 193 | 79 052 | 96 500 | 1 858 652 | |
| dep.rates | | | | | 79 052 | 110 218 | 1 858 652 | |
| $3,3,2$ | 1 019 868 | 3 546 614 | 36 225 840 | 136 062 633 | 123 330 | 154 172 | 4 344 685 | 133,173 |
| dep.rates | | | | | 123 330 | 175 403 | 4 344 685 | |

| Configuration with symmetric number of spares and color independent/dependent transition rates in $ew_3$ and $RepM3$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $C$ size | $\|SRG\|$ | | $\|RG\|$ | | $ESRG$ (Tan+Van) | | | |
| $\|C1\|,\|C2\|,\|C3\|$ | Tangible | Vanishing | Tangible | Vanishing | $\|ESRG\|$ | $\|Strong\|$ | $\|Exact\|$ | DSRG |
| $2,2+2$ | 44 915 | 131 482 | 1 055 314 | 3 433 358 | 43 307 | 52 509 | 173 689 | 46 896 |
| $2,2,2$ | 174 654 | 543 900 | 1 055 314 | 3 433 358 | 43 307 | 52 509 | 689 260 | 46 896 |
| dep.rates | | | | | 43 307 | 67 618 | 689 260 | 57 147 |
| $3,2+2$ | 129 895 | 413 617 | 7 736 126 | 27 382 639 | 81 815 | 101 982 | 532 883 | 89 360 |
| $3,2,2$ | 499 144 | 1 673 558 | 7 736 126 | 27 382 639 | 81 815 | 101 982 | 2 076 213 | 89 360 |
| dep.rates | | | | | 81 815 | 143 348 | 2 076 213 | 99 611 |
| $3,3,2$ | 1 116 480 | 3 969 892 | 39 705 528 | 151 977 288 | 126 798 | 161 896 | 4 843 938 | 139 219 |
| dep.rates | | | | | 126 798 | 269 711 | 4 843 938 | 162 469 |

**Table 1** Symbolic/Ordinary RG size of the FMS colored model and size of its unfolding for different configurations.

ability graph (SRG and ESRG). In a symbolic RG, multiple markings are lumped together into a single symbolic marking. SRG and ESRG have two different criteria for this marking aggregation.

GreatSPN can generate both the *ordinary* RG, equivalent to the RG of the unfolded model, and a more compact *symbolic* RG for any SN model. The latter exploits model symmetries and aggregates equivalent states. A CTMC can be derived both from the RG and from the SRG, on which both transient and steady state analysis can be performed. Extended and Dynamic SRG [9] (ESRG and DSRG) are also available, to efficiently deal with partially symmetric systems. Finally a simulator allows to compute estimates of steady state performance measures (with confidence intervals). The simulator supports both the plain colored marking representation and the symbolic one (which can improve the efficiency of future event list handling [22, 12]).

In Table 1 it is possible to compare the size of the RG, SRG, ESRG and DSRG for the test cases. The number of states in the RG is derived directly from the information contained in the SRG, but it can also be derived by direct generation from the model, at the price of higher cost in both time and space, or from the unfolded model. The state space reduction becomes relevant as the cardinality of class $C$ grows. The size of the SRG is 17% of the RG size for the case $|C1| = |C2| = |C3| = 2$, the 6% for the case $|C1| = 3, |C2| = |C3| = 2$, and 3% for the case $|C1| = |C2| = 3, |C3| = 2$.

Observe that a coarser partition into static subclasses gives better results in terms of state space reduction. Indeed when transition rates do not depend on the color, the two static subclasses $C2$ and $C3$ can be merged and the SRG shrinks further: in the cases $|C1| = 2, |C2| = 4$ and $|C1| = 3, |C2| = 4$, where two static subclasses of cardinality two have been merged, the SRG size is respectively 4% and 1.5% of the corresponding RG.

In systems where there are (partial) symmetries which cannot be exploited by SRG, it is still possible to take advantage of this symmetries by means of two solvers: The Extended SRG and the Dynamic SRG. The FMS model of Fig.13 has three static subclasses. The elements in the first subclass are routed to machine $m_2$ after leaving $m_1$, while the others are routed to $m_3$. The subclasses $C_2$ and $C_3$ are needed only in some configuration where the rates of certain transitions (in our example $ew_3$ and $RepM3$) are different for elements in classes $C_2$ and elements in the other classes. By executing the ESRG module on the FMS model the algorithm automatically detects and exploits partial symmetries. The generation of the CTMC according to the ESRG method comprises two steps. First a graph is built that overaggregates the states. Then a refinement step derives a lumped CTMC based either on *strong lumpability* or *exact lumpability* criteria [9]. The choice of the type of refinement depends on the performance indices the user wants to compute. Exact lumpability allows to retrieve the probability of detailed states, because it ensures equiprobability of the states in the same aggregate. In Table 1 the number of states generated with the ESRG algorithm are shown for some configurations. In particular observe that the size of the structure generated in the first phase of the ESRG derivation is much smaller than the size of the SRG for the same model (the number of states of the ESRG structure includes both tangible and vanishing markings). However, the refinement with the exact lumpability condition results in a size that is close at the SRG one. When the refinement is performed using the strong lumpability condition, the size of the refined lumped CTMC is only slightly larger than that unrefined one. In the strong lumpability case however only some color dependent performance indices can be computed.

Another possibility is to apply the DSRG solver. This aggregates the state space yielding a lumped CTMC satisfying the *exact lumpability* condition. The model specification in this case must be completely symmetric (no static subclasses partition of color classes, no guards, symmetric initial marking) while the asymmetries can be expressed in a separate file where it is possible to indicate, for each transition, restrictions on the colors that can be bound to each transition variable. In Table 1 the DSRG size for the model with equal number of spares for both machines and with or without color dependent transition rates is shown. The type of lumpability condition makes it possible to compute color dependent performance indices based on the information contained in the DSRG structure.

Measures can be defined for colored nets in the same way as for the uncolored ones: their definition can be independent of the color (average number of tokens in a place regardless of their color, or overall throughput of a transition corresponding to the sum of the throughput of any instance of that transition) or be color dependent. An interesting feature of the SRG is that, despite the relevant state space aggrega-

tion, it allows one to derive the same performance indices that could be computed on the much larger RG. The same is true for the ESRG with exact lumpability refinement and for the DSRG (which also ensures exact lumpability). The model checking facilities of GreatSPN currently require that colored models have to be first unfolded for the analysis to be performed.

| Throughput load | | | Utilization | | | | | | | Pr(Down) | | T queue |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | M1 | | | M2 | M3 | | | M2 | M3 | |
| C1 | C2 | C3 | C1 | C2 | C3 | | C1 | C2 | C3 | | | M2 |
| Different number of spares, uniform $ew_3$ rate. | | | | | | | | | | | | |
| 0.1906 | 0.2024 | 0.2024 | 0.1588 | 0.1686 | 0.1686 | 0.1059 | 0.1003 | 0.1065 | 0.1065 | 3.4e-7 | 0.0418 | 0.0310 |
| 0.2768 | 0.1963 | 0.1963 | 0.2306 | 0.1636 | 0.1636 | 0.1537 | 0.1457 | 0.1033 | 0.1033 | 2.1e-6 | 0.0538 | 0.0640 |
| 0.2678 | 0.2845 | 0.1896 | 0.2232 | 0.2371 | 0.1580 | 0.1488 | 0.1839 | 0.1952 | 0.1952 | 2.1e-6 | 0.0658 | 0.0624 |
| Different number of spares, color dependent $ew_3$ rate. | | | | | | | | | | | | |
| 0.1899 | 0.2005 | 0.2016 | 0.1583 | 0.1671 | 0.1680 | 0.1043 | 0.0999 | 0.1179 | 0.1061 | 3.5e-7 | 0.0444 | 0.0310 |
| 0.2757 | 0.1945 | 0.1955 | 0.2298 | 0.1621 | 0.1629 | 0.1532 | 0.1451 | 0.1144 | 0.1029 | 2.1e-6 | 0.0565 | 0.0639 |
| 0.2661 | 0.2812 | 0.1884 | 0.2218 | 0.2343 | 0.1570 | 0.1478 | 0.1401 | 0.1654 | 0.0992 | 2.2e-6 | 0.0701 | 0.0622 |
| Equal number of spares, uniform $ew_3$ rate. | | | | | | | | | | | | |
| 0.1953 | 0.2075 | 0.2075 | 0.1628 | 0.1729 | 0.1729 | 0.1085 | 0.1028 | 0.1092 | 0.1092 | 2.8e-5 | 0.0092 | 0.0314 |
| 0.2853 | 0.2025 | 0.2025 | 0.2378 | 0.1688 | 0.1688 | 0.1521 | 0.1502 | 0.1066 | 0.1066 | 1.1e-4 | 0.0136 | 0.0652 |
| 0.2776 | 0.2951 | 0.1967 | 0.2313 | 0.2459 | 0.1639 | 0.1542 | 0.1461 | 0.1553 | 0.1035 | 1.3e-4 | 0.0188 | 0.0638 |
| Equal number of spares, color dependent $ew_3$ rate. | | | | | | | | | | | | |
| 0.1949 | 0.2059 | 0.2070 | 0.1624 | 0.1716 | 0.1725 | 0.1083 | 0.1026 | 0.1211 | 0.1089 | 2.9e-5 | 0.0101 | 0.0313 |
| 0.2846 | 0.2009 | 0.2020 | 0.2372 | 0.1674 | 0.1683 | 0.1581 | 0.1498 | 0.1182 | 0.1063 | 1.2e-4 | 0.0148 | 0.0651 |
| 0.2764 | 0.2923 | 0.1959 | 0.2303 | 0.2436 | 0.1632 | 0.1536 | 0.1455 | 0.1719 | 0.1031 | 1.3e-4 | 0.0208 | 0.0637 |

**Table 2** Performance indices of the FMS colored model. Results grouped in set of three lines for workloads 2,2,2, 3,2,2 and 3,3,2

Table 2 shows some measures of interest computed using the GreatSPN solvers exploiting the model (partial) symmetries. These measures include system throughput (partitioned on the static subclasses of $C$), machines utilization, and probability for machines $m_2$ and $m_3$ to be unavailable due to a breakdown. Applying Little's formula we also derive the average time spent in $m_2$ queue (obtained as the ratio between the average number of customers in queue and the throughput of machine $m_2$). The measures are shown for the configurations (a) 2,2,2, (b) 3,2,2 and (c) 3,3,2, Each of these three configurations are tested in four different scenarios, that have/have not the same number of spares for each machine, as well as in case of uniform or color dependent rates of the $ew_3$ and $RepM3$ transitions.

The analysis of the model for an increasing number of elements in class C is limited by the state space size that grows considerably despite the application of techniques able to exploit the model behavioral symmetries. It is however possible to estimate the measures of interest through the SSN simulator.

Table 3 reports system throughput values, utilization of machine $m_1$, and probability of the repairman being in waiting status ($\#Ready == 1$, i.e. one token in place $Ready$) or working ($\#SpareRapairing == 1$, i.e. one token in place $SpareRepairing$). Each of these four measures are computed for 6 different sizes of the color classes, reported as triplets in the first column. The simulator can work with the same sym-

bolic marking representation used for the SRG, so that the future event list size (shown in the last column of Table 3) does not grow significantly when the number of colors in the color classes increases.

| Color class size | Performance indices | Point estimate | Confidence interval | time (sec) | average event list size |
|---|---|---|---|---|---|
| 3,3,2 | X(load) | 0.7355 | 0.7306, 0.7406 | 31 | 4.3783 |
| | U(M1) | 0.6134 | 0.6082, 0.6186 | | |
| | E(#Ready) | 0.6398 | 0.6273, 0.6524 | | |
| | E(#SpareRep) | 0.3003 | 0.2888, 0.3116 | | |
| 3,8,2 | X(load) | 0.9683 | 0.9580, 0.9788 | 165 | 4.9284 |
| | U(M1) | 0.8108 | 0.8017, 0.8200 | | |
| | E(#Ready) | 0.5509 | 0.5351, 0.5667 | | |
| | E(#SpareRep) | 0.3751 | 0.3606, 0.3895 | | |
| 3,8,8 | X(load) | 1.0794 | 1.0671, 1.0920 | 172 | 5.1236 |
| | U(M1) | 0.8970 | 0.8879, 0.9064 | | |
| | E(#Ready) | 0.5006 | 0.4836, 0.5177 | | |
| | E(#SpareRep) | 0.4158 | 0.4001, 0.4312 | | |
| 5,8,8 | X(load) | 1.0725 | 1.0623, 1.0850 | 180 | 5.2520 |
| | U(M1) | 0.9012 | 0.8923, 0.9103 | | |
| | E(#Ready) | 0.4811 | 0.4659, 0.4964 | | |
| | E(#SpareRep) | 0.4304 | 0.4167, 0.4439 | | |
| 8,8,8 | X(load) | 1.0957 | 1.0827, 1.1089 | 283 | 5.4395 |
| | U(M1) | 0.9219 | 0.9129, 0.9311 | | |
| | E(#Ready) | 0.4765 | 0.4582, 0.4950 | | |
| | E(#SpareRep) | 0.4340 | 0.4174, 0.45031 | | |
| 10,10,10 | X(load) | 1.1096 | 1.0975, 1.1218 | 291 | 5.4807 |
| | U(M1) | 0.9226 | 0.9142, 0.9310 | | |
| | E(#Ready) | 0.4383 | 0.4217, 0.4550 | | |
| | E(#SpareRep) | 0.4687 | 0.4534, 0.4838 | | |

**Table 3** Performance indices obtained with the GreatSPN SSN simulator (Confidence level 95% accuracy 2%).

## 7 Literature review

GreatSPN3.0 has many features that can find in similar software packages for the analysis of Petri-net based models. A full comparison of these tools would require a chapter on its own. In this section we only provide a brief overview of a few other tools that share some of the features of GreatSPN3.0 with hints on similarities and differences. Two characteristics that are unique to GreatSPN and that will not be listed explicitly as differences are the availability of a $CSL^{TA}$ model-checker and of solution techniques based on symmetries for colored Petri nets. Vice-versa, some of the tools listed below support compositionality through hierarchical models: this is a feature that is not present in GreatSPN.

*SPNP*. The software package SPNP ( [28] was developed in the 90's at Duke University, by the group of Trivedi, and it has evolved over the years to account for new research results in the field. SPNP basic formalism is that of SRNs, which incorporate several structural extensions to GSPNs such as marking dependencies (as marking dependent arc cardinalities and guards) and allow reward rates to be associated with each marking. Type of measures that can be computed are steady-state, transient, cumulative transient, time-averaged and up-to-absorption. A discrete-event simulator is available for both SRN and its non-Markovian extension. Limited support is provided for qualitative analysis of models, which is partially due to the choice of using a powerful text-based modelling language, for which a smaller number of qualitative analysis techniques are available. SRN definition was done in a C-like language, but in 2000 a Tcl/Tk-based graphical interface was added to reduce the need for the modeler to express the model in a purely textual form. SPNP graphical interface can also be used to draw and simulate fluid Petri nets.

*Möbius*. The tool Möbius [19] is an extensible dependability, security, and performance modelling environment for studying large-scale discrete-event systems. It supports multiple model formalisms which allow the modeller to represent each part of a system in the formalism that is most appropriate for it. Among the available formalisms it supports Stochastic Activity Networks (SANs) [35] a superclass of GSPNs in which the primitive *Input* and *Output* gates allow one to specify complex transition behaviours by general functions written in a C-like language. Like GreatSPN, Möbius provides multiple solution techniques. It support time and space efficient discrete-event simulation as well as numerical solutions based on compact MDD representation of the state space. Möbius is probably the most mature tool for Petri net (a commercial version is available as well). With respect to GreatSPN3.0 it has better features for stochastic simulation, but it has a limited support to structural and qualitative analysis.

*TimeNET*. The software package TimeNET [38], now at version 4.3, was developed starting back in 1995 at the Technische Universität of Ilmenau, as a successor of the software DSPNexpress, which was partly inspired by GreatSPN. The main focus of TimeNET is an efficient unified solution of DSPN and GSPN nets. Steady state and transient analysis techniques include either exact numerical solutions, approximate solutions and simulations. Firing delays of non-exponential transitions may have an arbitrary distribution. The graphical user interface, initially developed in Motif and then rewritten in Java, supports colored stochastic Petri nets as well as Markov chains, and is designed to be extensible to graph-like modeling formalisms. TimeNET provides more support for general distribution than GreatSPN3.0, but it does not include a complete qualitative analysis as provided by the CTL model-checker of GreatSPN.

*Snoopy-Marcie-Charlie*. The University of Cottbus has developed a suite of tools for the analysis of qualitative and stochastic properties of Petri nets which shares many of the objectives of GreatSPN3.0. Snoopy[25] is a java-based graphical interface which includes token animation. Marcie [26] is a set of analysis algorithms for various forms of stochastic Petri nets. These algorithms implement CTL and CSL

model-checking as well as CTMC solution and stochastic simulation of the net. CTL model checker is very efficient, based on a specific class of decision diagrams called IDD which are particularly efficient for Petri nets. Marcie is a command line tool, but it can be called through Charlie [27], which is an "extensible" interface for solvers. Charlie computes standard structural and behavioural properties of Petri nets, complemented by non-symbolic CTL and LTL model checking. Through the use of plug-in Charlie can work as interface for the command line solvers included in Marcie.

The suite of tools has been developed over numerous years and it is now very rich. It is not always easy to find the right solver to use and how to use it, so the wide choice of available solvers is not very easy to use. We believe that the Cottbus suite is facing a situation similar to that we experienced in GreatSPN before deciding to re-write the GUI and to link from the GUI all the available solvers. The Cottbus suite puts emphasis on structural analysis and on the richness of Petri net extension the tool is able to deal with. Great attention is devoted to stochastic simulation of Petri nets of biological system. Features available in GreatSPN and not in the Cottbus suite include $CSL^{TA}$ model-checking and efficient solution of colored models.

*Smart*. Smart[36] or *Stochastic Model-checking Analyzer for Reliability and Timing* is a software package providing command-line environment for logic and probabilistic analysis of complex systems. Its main input formalism are Stochastic Petri nets and both discrete-time and continuous time Markov chains. For the analysis of logical behaviour, both explicit and symbolic state-space generation techniques are available. In the new release, currently under development, all the symbolic algorithms will be based on Meddly library as in GreatSPN. For the study of stochastic and timing behaviour, sparse-storage, symbolic and Kronecker numerical solution approaches are available when the underlying process is a Markov chain. Discrete-event simulation is also provided.

*APNNtoolbox* The APNNtoolbox [8] has been developed at the University of Dortmund as an open toolset around a common exchange interface denoted as the Abstract Petri Net Notation (APNN). The tool provides support for stochastic Petri net, including support for hierarchies and for a limited form of colours. The solvers in the APNNtoolbox are focused on state-space based analysis methods, where state-space explosion is dealt with through Kronecker representation. The toolbox also includes a graphical user interface (APNNed) to draw the net and call the solvers.

*QPME* QPME [31] is a tool developed initially at the University of Darmstadt, and currently maintained at the University of Würzburg. It is devoted to Stochastic Petri nets with the extension of queueing places for which the tool provides discrete event simulation.

*Oris* The Oris tool [13] has been developed at the University of Florence to deal with Timed and stochastic Petri nets. The delay associated with transitions can either be a non deterministic value, between a pair of min-max boundaries, or stochastic over a finite/infinite interval, thus subsuming also classical stochastic Petri nets. The tool is equipped with a graphical interface for net specification and for the display of the

analysis results. Oris is oriented at the analysis of non-Markovian system, for which it provides the most advanced solvers currently available.

## 8 Future work

A future work list for GreatSPN strictly depends on what will be the research results in the performance evaluation field in the next years, given the willingness of keeping GreatSPN always at pace with the most useful research advances. Such a list is difficult to write, but there are nevertheless a few features of the current graphical interface and associated solvers that are already planned for a (hopefully close) future. The first enhancement is to develop a CTL model-checker for colored models to improve the analysis capabilities of GreatSPN3.0 already made unique by the use of the $CSL^{TA}$ model checker. It could be interesting if such model-checker could work directly on the symbolic reachability graph. Another approach to solve this problem is to unfold a colored Petri net (like a SWN) into its uncolored (GSPN) equivalent and then perform an (uncolored) model-checking. Another point that deserves more attention in the tool is the definition of the color-dependent performance indices that would complement in an extremely useful manner the efficient solution techniques based on symmetries for colored Petri nets already implemented in GreatSPN3.0. Finally, compositionality of Petri net models is clearly a desired feature, that would make it easier to draw complex hierarchical models by separating the logic into multiple nets, supporting both top-down and bottom-up approaches. In GreatSPN3.0 net composition can be performed through a command line program called algebra, yet to be integrated in the GUI, which implements a parallel operator similar to the parallel operator in process algebra (in CSP style) based on transition superposition; additionally, the algebra program also implements place superposition. Upgrading this feature to the level of the invocation of the other analysis capabilities of the tool through the new GUI is obviously an enhancement that we would like to develop as soon as possible.

## References

[1] Ajmone Marsan M, Balbo G, Ciardo G, Conte G (1984) A software tool for the automatic analysis og generalized stochastic Petri net models. In: Proceedings. 1st International Conference on Modeling Techniques and Tools for Performance Analysis, INRIA, Paris, France, pp 243–258

[2] Ajmone Marsan M, Conte G, Balbo G (1984) A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Transactions on Computer Systems 2:93–122

[3] Ajmone Marsan M, Balbo G, Conte G, Donatelli S, Franceschinis G (1995) Modeling with Generalized Stochastic Petri Nets. J. Wiley, New York, NY,

USA, URL http://www.di.unito.it/~greatspn
[4] Ajmone Marsan M, Balbo G, Conte G (2000) The early days of GSPNs. In: Performance Evaluation: Origins and Directions, Springer-Verlag, London, UK, UK, pp 505–512
[5] Amparore E, Beccuti M, Donatelli S (2014) (stochastic) model checking in greatspn. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 8489 LNCS:354–363
[6] Amparore EG (2015) Reengineering the editor of the GreatSPN framework. In: Moldt D, Rölke H, Störrle H (eds) Petri Nets and Software Engineering. International Workshop, PNSE'15, Brussels, Belgium, June 22-23, 2015. Proceedings, CEUR-WS.org, CEUR Workshop Proceedings, vol 1372, pp 153–170
[7] Amparore EG, Donatelli S (2010) MC4CSL$^{TA}$: an efficient model checking tool for CSL$^{TA}$. In: International Conference on Quantitative Evaluation of Systems, IEEE Computer Society, Los Alamitos, CA, USA, pp 153–154
[8] APNNtoolbox (2015) APNNtoolbox webpage. http://ls4-www.cs.tu-dortmund.de/APNN-TOOLBOX/
[9] Baarir S, Beccuti M, Dutheillet C, Franceschinis G, Haddad S (2011) Lumping partially symmetrical stochastic models. Performance Evaluation 68(1):21–44
[10] Balbo G, Chiola G (1989) Stochastic Petri net simulation. In: Proceedings of the 21st Conference on Winter Simulation, ACM, New York, NY, USA, WSC '89, pp 266–276
[11] Balbo G, Beccuti M, De Pierro M, Franceschini G (2011) Computing first passage time distributions in stochastic well-formed nets. SIGSOFT Softw Eng Notes 36(5):7–18
[12] Beccuti M, Franceschinis G (2012) Efficient simulation of stochastic well-formed nets through symmetry exploitation. In: Proceedings - Winter Simulation Conference, 2012.
[13] Bucci G, Carnevali L, Ridi L, Vicario E (2010) Oris: a tool for modeling, verification and evaluation of real-time systems. International Journal on Software Tools for Technology Transfer 12(5):391–403
[14] Buchholz P, Ciardo G, Donatelli S, Kemper P (2000) Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. INFORMS Journal on Computing 12(3):203–222
[15] Chiola G (1985) A Software package for the analysis of Generalized Stochastic Petri Net models. In: International Workshop on Timed Petri Nets, IEEE Computer Society, Washington, DC, USA, pp 136–143
[16] Chiola G (1988) Compiling techniques for the analysis of stochastic Petri nets. In: Proceedings of the 4th Int. Conference on Modeling Techniques and Tools for Computer Performance Evaluation, AFCET, pp 11–24
[17] Chiola G, Ajmone Marsan M, Balbo G, Conte G (1993) Generalized stochastic petri nets: a definition at the net level and its implications. Software Engineering, IEEE Transactions on 19(2):89–107

[18] Chiola G, Dutheillet C, Franceschinis G, Haddad S (1993) Stochastic well-formed colored nets and symmetric modeling applications. Computers, IEEE Transactions on 42(11):1343–1360

[19] Clark G, Courtney T, Daly D, Deavours D, Derisavi S, Doyle J, Sanders W, Webster P (2001) The mobius modeling tool. In: Petri Nets and Performance Models, 2001. Proceedings. 9th International Workshop on, pp 241–250

[20] Clarke EM, Emerson EA (1982) Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen D (ed) Logics of Programs, Workshop, Yorktown Heights, New York, May 1981, Springer, Lecture Notes in Computer Science, vol 131, pp 52–71

[21] Donatelli S, Haddad S, Sproston J (2009) Model checking timed and stochastic properties with CSL$^{TA}$. IEEE Transactions on Software Engineering 35(2):224–240

[22] Gaeta R (1996) Efficient discrete-event simulation of colored petri nets. IEEE Trans Software Eng 22(9):629–639

[23] German R (2000) Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets. John Wiley & Sons, Inc., New York, NY, USA

[24] GreatSPN (2015) GreatSPN webpage. `http://www.di.unito.it/~greatspn/index.html`

[25] Heiner M, Herajy M, Liu F, Rohr C, Schwarick M (2012) Snoopy - A unifying petri net tool. In: Application and Theory of Petri Nets - 33rd International Conference, PETRI NETS 2012, Hamburg, Germany, June 25-29, 2012. Proceedings, pp 398–407

[26] Heiner M, Rohr C, Schwarick M (2013) MARCIE - model checking and reachability analysis done efficiently. In: Colom JM, Desel J (eds) Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings, Springer, Lecture Notes in Computer Science, vol 7927, pp 389–399

[27] Heiner M, Schwarick M, Wegener J (2015) Charlie - an extensible petri net analysis tool. In: Devillers RR, Valmari A (eds) Application and Theory of Petri Nets and Concurrency - 36th International Conference, PETRI NETS 2015, Brussels, Belgium, June 21-26, 2015, Proceedings, Springer, Lecture Notes in Computer Science, vol 9115, pp 200–211

[28] Hirel C, Tuffin B, Trivedi KS (2000) Spnp: Stochastic petri nets. version 6.0. In: Proceedings of the 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, Springer-Verlag, London, UK, UK, TOOLS '00, pp 354–357

[29] ISO/IEC (2011) Standard ISO/IEC 15909-2:2011, Systems and software engineering – High-level Petri nets. `http:www.iso.org/iso/catalogue_detail?csnumber=43538`, accessed: 2015-08-05

[30] Jensen K, Wirth N (1975) PASCAL user manual and report. Springer-Verlag, Inc, New York, NY, USA

[31] Kounev S, Spinner S, Meier P (2010) Qpme 2.0 - a tool for stochastic modeling and analysis using queueing petri nets. In: Sachs K, Petrov I, Guerrero P (eds)

From Active Data Management to Event-Based Systems and More, Lecture Notes in Computer Science, vol 6462, Springer Berlin Heidelberg, pp 293–311

[32] Molloy M (1982) Performance analysis using stochastic petri nets. Computers, IEEE Transactions on C-31(9):913–917

[33] Molloy MK, Riddle P (1986) The Stochastic Petri Net Analyzer system design tool for Bit-mapped workstations. Computer Science Department, University of Texas at Austin

[34] Petri C (1962) Kommunikation mit Automaten. PhD thesis, Schriften des Institutes fr Instrumentelle Matematik, Bonn

[35] Sanders W, Meyer J (2001) Stochastic Activity Networks: Formal Definitions and Concepts. In: Brinksma E, Hermanns H, Katoen JP (eds) Lectures on Formal Methods and PerformanceAnalysis, Lecture Notes in Computer Science, vol 2090, Springer Berlin Heidelberg, pp 315–343

[36] Smart (2015) SMART webpage. `http://www.cs.ucr.edu/~ciardo/SMART`

[37] Teruel E, Franceschinis G, De Pierro M (2003) Well-defined generalized stochastic petri nets: A net-level method to specify priorities. IEEE Transactions on Software Engineering 29(11):962–973

[38] Zimmermann A (2012) Modeling and evaluation of stochastic petri nets with timenet 4.1. In: Gaujal B, Jean-Marie A, Jorswieck EA, Seuret A (eds) 6th International ICST Conference on Performance Evaluation Methodologies and Tools, Cargese, Corsica, France, October 9-12, 2012, ICST/IEEE, pp 54–63